CS 406 Semester Project (Tentative)

Spring 2003

March 12, 2003

Abstract

The project is specified briefly, followed by a listing of the artifacts which are its result. Though some requirements are explicit one may find that their interaction produces other implicit requirements whose discovery is the task of the Requirements Activities. This document does not contain a complete specification of the project, and the specification given herein may 'evolve' over the course of the semester. This is the case with any non-trivial project. Note that the primary activity undertaken in the project is thought. Don't be tempted to jump too early into creation of the required artifacts; the artifacts do not create understanding of the project, the understanding of the project helps create the artifacts – which are only notes and reminders.

1 Project Overview

The CS 406 Semester project for Spring 2003 is the following: The development (in all of its many phases) of the Generic Document System (GDS)

1.1 Business Scenario (fictitious)

Troubled by the lack of commercially available software with all of the required functionality, businesses have demanded extensibility in the systems they purchase so that they may contract the construction of add-on modules (or "plugins") to add needed features.

To capture this market, we have need of a software product which is 100% adaptable – which will allow us to market a large library of add-on modules. The business model is "pay-by-feature". Thus a user selects the features at installation, and is billed accordingly.

1.2 Project details

The specific project you are to implement is a document preparation system. (simillar to Adobe Pagemaker, or Microsoft Publisher). A document is essentially a collection of objects – text areas, pictures, etc. – and their layout. Given the motivation above, the goal is to implement a "generic" document preparation system – one designed around the concept of plug-ins and 'pay by feature'. This presents a Software Engineering challenge in that the software must be designed for extensibility to allow an unknown set of future extentions.

The following are the known requirements of the system:

- GUI, What You See Is What You Get

- The application (GDS) itself contains only standard menu functions[1]: *Open, Close, Save, Exit, Cut, Copy, Paste, Help, and window management options (e.g. bring to top, etc.).*
- The GDS supports 'multiple document interface' or several documents open simultaneously in the application.
- The GDS allows basic manipulation of object layout via the mouse.

The following plugins are required as part of the development effort:

- a Text Area plugin. A text area is a rectangular section of a document with an associated font (size, color, etc). A text area allows the user to type the desired text into the text area, and displays it using the chosen font. The plugin should create a menu item in the "Insert" menu with the title "Text Area". The "hot-key" associated with the "Insert→Text Area" operation is CTRL+T[2].
- an Image plugin. An image is a rectangular section of a document that displays an image loaded from a file. The plugin should create a menu item in the "Insert" menu with the title "Image (from File)". The hot-key associated with the "Insert→Image (from File)" operation is CTRL+I
- an Undo plugin. This plugin will add a menu item to the "Edit" menu with the title "Undo". The associated hot-key is CTRL+Z. The Undo option should be able to undo no less than 10 operations.
- a 'remote collaboration' plugin. This allows other users to connect to the GDS, and edit documents that are currently open, and 'shared'. Menu and hot-key additions for this feature are at the discretion of the developer. The plugin should allow both server and client features. While there are no specific behaviors required, the remote collaboration feature should be "usable", "convienient", and should manage concurrent operations in a "fair" way. Network operations may be optimized toward a small LAN environment.

2 Notes

- Don't procrastinate. This project sounds easy, but there are many subtle interactions that will require mechanisms to support them.
- Being a GUI application, you will probably use Java, or something forWindows. If you use something else, be sure that you will be able to load plugins. (For Java, there's always reflection, and for Windows, there's Dynamically Linked Libraries (.dll))

3 Deliverables/Artifacts

As any non-trivial software project is too large to keep entirely in your memory. The following artifacts will serve both as aides to your development effort, and as the criteria upon which you will be evaluated.

3.1 Requirements Specification (Ch. 4)

3.1.1 Deliverables

There are several deliverables for the Requirements phase:

[1]No part of the plugin functionality should be in the application, the application only provides support for plugins via communication protocols, etc.

[2]Depending on the platform, CTRL may not be feasible, so substitutions may be made.

- Requirements Document – a rather informal document listing the explicit and implied requirements gathered from the user. The Requirements Document is a requirements gathering and analysis artifact that is not intended for the user. Your requirements document should contain the use cases you have identified for the GDS.
- Prototype – of the application. An application with some of the features implemented, some features faked, and some features omitted. The purpose of the prototype is to allow the user to see (concretely) what you're planning on building for them, so changes can be made before money is spent in development.
- Requirements Specification Document – This is a formal document which details the requirements of the software that will be developed. It is usually viewed as a binding legal contract between the developer and the client. The Specification will include functional requirements (such as required features) and non-functional requirements (such as the target platform, or timing requirements.) An outline for the document will be provided shortly. Keep in mind that a formal document strives to be as unambiguous, clear, complete, and brief as possible. A helpful tip for specifying requirements of software features is to describe the feature in terms of input and output, or in terms of pre- and post- conditions.
- Project Management Plan – now that the requirements are fixed, you should be able to make an initial guess as to how you will go about building the system. The PMP should include the lifecycle model you plan to use, a schedule for the milestones you define, team organization, member responsibilities, and any other details you feel will help guide your progress. (This is by no means final, and its *accuracy* will not affect your grade; you should, however, try to estimate accurately as it will be a useful tool to your team.)
- Peer Review Report – As part of requirements specification, we will ask you to review the Requirements Specification Document prepared by another team. You should note items that you feel are exceptional, as well as those which need revision. As always, your reasons should accompany your suggestions. (*You will likely find items you omitted during this process.*)

3.1.2 Evaluation

The Requirements Specification Document should be formal, complete, consistent, and unambiguous. All requirements identified in the Requirements Document should be specified here. Ambiguous vs. non-ambiguous can be illustrated as follows: "Then it must fit a curve to the data" vs. "Then a 3rd order polynomial curve is fit to the data using Least-Squares approximation". Also beware of inherently ambiguous wordsa and phrases such as 'involving' or 'having to do with'.

3.2 Design (Ch. 9)

3.2.1 Deliverables

There are two phases to design, however the results of these will be lumped into a single design document in two passes. (Document outline to be provided)

- Pass 1: Architectural Design – A decomposition of the application into subsystems and components. The interactions between these subsystems and components should be clear. UML Package diagrams may be an effective illustration here. You should include an interaction diagram for each use case (showing the subsystem-level interaction needed to support the use case). For each use case, you will also develop a test plan.
- Pass 2: Detailed Design – To the document from pass 1, the following will be added: further decomposition of the components and subsystems into modules. The module descriptions should include all

public members (public functions, variables, etc.), protected members that are explicitly required by the design, and private members that are necessary to understand the design. (e.g. Those related to the class' role). An interaction diagram for each use case will be developed detailing the inter-module interaction. For each module you will develop a test plan.

- Both: Design Decisions – Both pass 1 and 2 should include a section on design decisions that are made. Specifically, this should include explicit or crucial design decisions or philosophies, and the justification for why you have made your particular decision. This can include choice of algorithm, data structures, Design Patterns, inter-module communication, etc.
- Peer Review Report – A peer review, similar to that for Requirements

3.2.2 Evaluation

Evaluation of the Design Document will be with respect to completeness and correctness. The design should be cross-referenced to the requirements in the Requirements Specification Document. The document should be written unambiguously, using UML diagrams as illustrations (not in lieu of text). The test plans should satify appropriate criteria (coverage measures, Boundary Values, Equivalence Classes, etc.), and your design decisions should be clearly described, including their justification.

3.3 Implementation

3.3.1 Deliverables

- Source Code (including instructions and Makefile, or scripts to build)
- Test Code (from test plan for modules)
- Necessary changes to Design document
- An appendix (Appendix A) in the Design Document detailing team member responsibilities.

3.3.2 Evaluation

Adherence to the design document and [module] test plan.

3.4 Integration

3.4.1 Deliverables

- Source Code as per Implementation
- Test Code (Product test code)
- Necessary changes to Design document
- User's Manual
- Peer Review Report

3.4.2 Evaluation

The application will be tested for adherence to the requirements, completeness, and correctness. (robustness, reliability, etc.)

The Test code should implement the test plan. The design document should be up-to-date with the code. The user's manual should describe for the user how to use the features.

4 Schedule

Upon the project deadline, teams will submit the deliverables. The deliverables will then be sent to the peer-review partner. Peer review will be due 1 week after the project deadline, and will be distributed to the peer-review partner. Teams may then take two days to act upon peer-review comments and resubmit before evaluation.

Tentative Deadlines and Weights

Deliverable	Due Date	Weight	
Peer Review Reports	1 week after due dates	5% overall	
Prototype	Wednesday 2/5/03	1%	
Requirements Document	Monday 2/17/03	1%	
Requirements Specification	Monday 3/3/03	25%	
Project Management Plan	Monday 3/3/03	8%	
Design Part 1	Wednesday 3/30/03	15%	*Merged with Part 2
Design Part 2	Monday 3/30/03	15%	*Moved up a day
Implementation (Code/Tests)	Friday 4/11/03	10%	
Integration (Code/Tests)	Friday 4/18/03	10%	
User's Manual	Friday 4/18/03	10%	